Struct Address

Objective

- Learn basics of data structures
- Learn how memory may be padded within data structures

Review Basics

Here is the basic use of data structures in C:

```
// Declare data structure in C using typedef
typedef struct {
  int i;
  char c;
  float f;
} my_struct_t;

// Pass data structure as a copy
void struct_as_param(my_struct_t s) {
  s.i = 0;
  s.c = 'c';
}

// Pass data structure as a pointer
void struct_as_pointer(my_struct_t *p) {
  p->i = 0;
  p->c = 'c';
}

// Zero out the struct
void struct_as_pointer(my_struct_t *p) {
  memset(p, 0, sizeof(*p));
}
```

Padding

1. Use the struct below, and try this sample code
   - Note that there may be a compiler error in the snippet below that you are expected to resolve on your own
   - Struct should ideally be placed before the main() and the `printf()` should be placed inside of the `main()`
   - You should use your SJ embedded board because the behavior may be different on a different compiler or the board

2. Now un-comment the `packed` attribute such that the compiler packs the fields together, and print them again.

```
typedef struct {
  float f1; // 4 bytes
  char c1;  // 1 byte
```

```
  float f2;
  char c2;
} /*__attribute__((packed))*/ my_s;

// TODO: Instantiate a struct of type my_s with the name of "s"
printf("Size : %d bytes\n"
       "floats 0x%p 0x%p\n"
       "chars  0x%p 0x%p\n",
       sizeof(s), &s.f1, &s.f2, &s.c1, &s.c2);
```

Note:

- Important: In your submission (could be comments in your submitted code), provide your summary of the two print-outs. Explain why they are different, and try to draw conclusions based on the behavior.